Ethereum Virtual Machine Opcodes

Updated 2021-07-29 (Added `CHAINID`, `SELFBALANCE` and `BASEFEE` opcodes).

This is intended to be a low level reference for the Ethereum Virtual Machine.
If you're trying to learn how to write smart contracts, check out the official Solidity documentation instead.

Resources

- Yellow Paper
- go-ethereum opcodes implementation
- EIP-150 opcode gas costs
- Solidity Block and Transaction Properties
- Layout of State Variables in Storage

Opcodes

Use this handy table to skip ahead to the opcode reference.

00	01	02	03	04	05	06	07	08	09	0A	0B	–	–	–	–
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	–	–
20	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	–	–	–	–	–	–	–
50	51	52	53	54	55	56	57	58	59	5A	5B	–	–	–	–
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	–	–	–	–	–	–	–	–	–	–	–
B0	B1	B2	–	–	–	–	–	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
F0	F1	F2	F3	F4	F5	–	–	–	–	FA	–	–	FD	–	FF

Overview

The Ethereum VM is a stack-based, big-endian VM with a word size of 256-bits and is used to run the smart contracts on the Ethereum blockchain.
Smart contracts are just like regular accounts, except they run EVM bytecode when receiving a transaction, allowing them to perform calculations and further transactions.
Transactions can carry a payload of 0 or more bytes of data, which is used to specify the type of interaction with a contract and any additional information.

Contract execution starts at the beginning of the bytecode.
Each opcode is encoded as one byte, except for the PUSH opcodes, which take a immediate value.
All opcodes pop their operands from the top of the stack and push their result.

Contract Creation

The data payload of a transaction creating a smart contract is itself bytecode that runs the contract constructor, sets up the initial contract state and returns the final contract bytecode.
ie. constructors are not present in the contract once deployed.

Contract Interaction

Typically contracts expose a public ABI, which is a list of supported ways a user can interact with a contract.
To interact with a contract, a user will submit a transaction carrying any amount of wei (including 0) and a data payload formatted according to the ABI, specifying the type of interaction and any additional parameters.
When the contract runs there are 4 main ways it handles data.

Call Data

This is the data associated with a transaction to a smart contract. It usually contains a 4-byte method identifier followed by serialized arguments.

See: `CALLDATALOAD`, `CALLDATASIZE`, `CALLDATACOPY`

Stack

The EVM maintains a stack of `uint256`s used to hold local variables, function call arguments and return addresses.
Distinguishing between return addresses and other variables is tricky.

See: PUSH1, DUP1, SWAP1, POP

Memory

Memory is an array of `uint8`s used to hold transient data while a contract is being executed.

It is not persisted across transactions.

See: MLOAD, MSTORE, MSTORE8

Storage

Storage is a persistent associative map, with `uint256`s as keys and `uint256`s as values.

All contract fields and mappings are saved in storage.

Storage fields can be inspected using `web3.eth.getStorageAt(address, key)`.

See: SLOAD, SSTORE

Opcodes

uint8	Mnemonic	Stack Input	Stack Output	Expression	Notes
00	STOP	-	-	STOP()	halts execution of the contract
01	ADD	a b	a + b	a + b	(u)int256 addition modulo 2**256
02	MUL	a b	a * b	a * b	(u)int256 multiplication modulo 2**256
03	SUB	a b	a - b	a - b	(u)int256 subtraction modulo 2**256
04	DIV	a b	a // b	a // b	uint256 division
05	SDIV	a b	a // b	a // b	int256 division
06	MOD	a b	a % b	a % b	uint256 modulus
07	SMOD	a b	a % b	a % b	int256 modulus
08	ADDMOD	a b N	(a + b) % N	(a + b) % N	(u)int256 addition modulo N
09	MULMOD	a b N	(a * b) % N	(a * b) % N	(u)int256 multiplication modulo N
0A	EXP	a b	a ** b	a ** b	uint256 exponentiation modulo 2**256
0B	SIGNEXTEND	b x	y	y = SIGNEXTEND(x, b)	sign extends x from (b + 1) * 8 bits to 256 bits.
0C	*Invalid*	-	-	-	-
0D	*Invalid*	-	-	-	-
0E	*Invalid*	-	-	-	-
0F	*Invalid*	-	-	-	-
10	LT	a b	a < b	a < b	uint256 comparison
11	GT	a b	a > b	a > b	uint256 comparison
12	SLT	a b	a < b	a < b	int256 comparison
13	SGT	a b	a > b	a > b	int256 comparison
14	EQ	a b	a == b	a == b	(u)int256 equality
15	ISZERO	a	a == 0	a == 0	(u)int256 is zero
16	AND	a b	a & b	a & b	256-bit bitwise and
17	OR	a b	a | b	a | b	256-bit bitwise or
18	XOR	a b	a ^ b	a ^ b	256-bit bitwise xor
19	NOT	a	~a	~a	256-bit bitwise not

1A	BYTE	i | x	y	y = (x >> (248 - i * 8)) & 0xFF	counting from most significant byte
1B	SHL	shift | value	value << shift	value << shift	256-bit shift left
1C	SHR	shift | value	value >> shift	value >> shift	256-bit shift right
1D	SAR	shift | value	value >> shift	value >> shift	int256 shift right
1E	*Invalid*	-	-	-	-
1F	*Invalid*	-	-	-	-
20	SHA3	offset | length	hash	hash = keccak256(memory[offset:offset+length])	keccak256
21	*Invalid*	-	-	-	-
22	*Invalid*	-	-	-	-
23	*Invalid*	-	-	-	-
24	*Invalid*	-	-	-	-
25	*Invalid*	-	-	-	-
26	*Invalid*	-	-	-	-
27	*Invalid*	-	-	-	-
28	*Invalid*	-	-	-	-
29	*Invalid*	-	-	-	-
2A	*Invalid*	-	-	-	-
2B	*Invalid*	-	-	-	-
2C	*Invalid*	-	-	-	-
2D	*Invalid*	-	-	-	-
2E	*Invalid*	-	-	-	-
2F	*Invalid*	-	-	-	-
30	ADDRESS	-	address(this)	address(this)	address of the executing contract
31	BALANCE	addr	address(addr).balance	address(addr).balance	address balance in wei
32	ORIGIN	-	tx.origin	tx.origin	transaction origin address
33	CALLER	-	msg.caller	msg.caller	message caller address
34	CALLVALUE	-	msg.value	msg.value	message funds in wei
35	CALLDATALOAD	i	msg.data[i:i+32]	msg.data[i:i+32]	reads a (u)int256 from message data
36	CALLDATASIZE	-	msg.data.size	msg.data.size	message data length in bytes
37	CALLDATACOPY	destOffset | offset | length	-	memory[destOffset:destOffset+length] = msg.data[offset:offset+length]	copy message data
38	CODESIZE	-	address(this).code.size	address(this).code.size	length of the executing contract's code in bytes
39	CODECOPY	destOffset | offset | length	-	memory[destOffset:destOffset+length] = address(this).code[offset:offset+length]	copy executing contract's bytecode
3A	GASPRICE	-	tx.gasprice	tx.gasprice	gas price of the executing transaction, in wei per unit of gas
3B	EXTCODESIZE	addr	address(addr).code.size	address(addr).code.size	length of the contract bytecode at addr, in bytes
3C	EXTCODECOPY	addr | destOffset | offset | length	-	memory[destOffset:destOffset+length] = address(addr).code[offset:offset+length]	copy contract's bytecode
3D	RETURNDATASIZE	-	size	size = RETURNDATASIZE()	Byzantium hardfork, EIP-211: the size of the returned data from the last external call, in bytes
3E	RETURNDATACOPY	destOffset | offset | length	-	memory[destOffset:destOffset+length] = RETURNDATA[offset:offset+length]	Byzantium hardfork, EIP-211: copy returned data
3F	EXTCODEHASH	addr	hash	hash = address(addr).exists ? keccak256(address(addr).code) : 0	Constantinople hardfork, EIP-1052: hash of the contract bytecode at addr

40	BLOCKHASH	`blockNumber`	`hash`	`hash = block.blockHash(blockNumber)`	only valid for the 256 most recent blocks, excluding the current one
41	COINBASE	-	`block.coinbase`	`block.coinbase`	address of the current block's miner
42	TIMESTAMP	-	`block.timestamp`	`block.timestamp`	current block's Unix timestamp in seconds
43	NUMBER	-	`block.number`	`block.number`	current block's number
44	DIFFICULTY	-	`block.difficulty`	`block.difficulty`	current block's difficulty
45	GASLIMIT	-	`block.gaslimit`	`block.gaslimit`	current block's gas limit

46	CHAINID	-	`chain_id`	`chain_id = {  1 // mainnet`
				`{  2 // Morden testnet (disused)`
				`{  2 // Expanse mainnet`
				`{  3 // Ropsten testnet`
				`{  4 // Rinkeby testnet`
				`{  5 // Goerli testnet`
				`{ 42 // Kovan testnet`
				`{ ...`
					Istanbul hardfork, EIP-1344: current network's chain id

47	SELFBALANCE	-	`address(this).balance`	`address(this).balance`	Istanbul hardfork, EIP-1884: balance of the executing contract in wei
48	BASEFEE	-	`base_fee`	`base_fee = BASEFEE()`	London hardfork, EIP-3198: current block's base fee
49	*Invalid*	-	-	-	-
4A	*Invalid*	-	-	-	-
4B	*Invalid*	-	-	-	-
4C	*Invalid*	-	-	-	-
4D	*Invalid*	-	-	-	-
4E	*Invalid*	-	-	-	-
4F	*Invalid*	-	-	-	-
50	POP	`_`	-	`POP()`	pops a (u)int256 off the stack and discards it
51	MLOAD	`offset`	`value`	`value = memory[offset:offset+32]`	reads a (u)int256 from memory
52	MSTORE	`offset` `value`	-	`memory[offset:offset+32] = value`	writes a (u)int256 to memory
53	MSTORE8	`offset` `value`	-	`memory[offset] = value & 0xFF`	writes a uint8 to memory
54	SLOAD	`key`	`value`	`value = storage[key]`	reads a (u)int256 from storage
55	SSTORE	`key` `value`	-	`storage[key] = value`	writes a (u)int256 to storage
56	JUMP	`destination`	-	`$pc = destination`	unconditional jump
57	JUMPI	`destination` `condition`	-	`$pc = cond ? destination : $pc + 1`	conditional jump if condition is truthy
58	PC	-	`$pc`	`$pc`	program counter
59	MSIZE	-	`size`	`size = MSIZE()`	size of memory for this contract execution, in bytes
5A	GAS	-	`gasRemaining`	`gasRemaining = GAS()`	remaining gas
5B	JUMPDEST	-	-		metadata to annotate possible jump destinations
5C	*Invalid*	-	-	-	-
5D	*Invalid*	-	-	-	-
5E	*Invalid*	-	-	-	-
5F	*Invalid*	-	-	-	-
60	PUSH1	-	`uint8`	`PUSH(uint8)`	pushes a 1-byte value onto the stack
61	PUSH2	-	`uint16`	`PUSH(uint16)`	pushes a 2-byte value onto the stack

					onto the stack
63	PUSH4	-	uint32	PUSH(uint32)	pushes a 4-byte value onto the stack
64	PUSH5	-	uint40	PUSH(uint40)	pushes a 5-byte value onto the stack
65	PUSH6	-	uint48	PUSH(uint48)	pushes a 6-byte value onto the stack
66	PUSH7	-	uint56	PUSH(uint56)	pushes a 7-byte value onto the stack
67	PUSH8	-	uint64	PUSH(uint64)	pushes a 8-byte value onto the stack
68	PUSH9	-	uint72	PUSH(uint72)	pushes a 9-byte value onto the stack
69	PUSH10	-	uint80	PUSH(uint80)	pushes a 10-byte value onto the stack
6A	PUSH11	-	uint88	PUSH(uint88)	pushes a 11-byte value onto the stack
6B	PUSH12	-	uint96	PUSH(uint96)	pushes a 12-byte value onto the stack
6C	PUSH13	-	uint104	PUSH(uint104)	pushes a 13-byte value onto the stack
6D	PUSH14	-	uint112	PUSH(uint112)	pushes a 14-byte value onto the stack
6E	PUSH15	-	uint120	PUSH(uint120)	pushes a 15-byte value onto the stack
6F	PUSH16	-	uint128	PUSH(uint128)	pushes a 16-byte value onto the stack
70	PUSH17	-	uint136	PUSH(uint136)	pushes a 17-byte value onto the stack
71	PUSH18	-	uint144	PUSH(uint144)	pushes a 18-byte value onto the stack
72	PUSH19	-	uint152	PUSH(uint152)	pushes a 19-byte value onto the stack
73	PUSH20	-	uint160	PUSH(uint160)	pushes a 20-byte value onto the stack
74	PUSH21	-	uint168	PUSH(uint168)	pushes a 21-byte value onto the stack
75	PUSH22	-	uint176	PUSH(uint176)	pushes a 22-byte value onto the stack
76	PUSH23	-	uint184	PUSH(uint184)	pushes a 23-byte value onto the stack
77	PUSH24	-	uint192	PUSH(uint192)	pushes a 24-byte value onto the stack
78	PUSH25	-	uint200	PUSH(uint200)	pushes a 25-byte value onto the stack
79	PUSH26	-	uint208	PUSH(uint208)	pushes a 26-byte value onto the stack
7A	PUSH27	-	uint216	PUSH(uint216)	pushes a 27-byte value onto the stack
7B	PUSH28	-	uint224	PUSH(uint224)	pushes a 28-byte value onto the stack
7C	PUSH29	-	uint232	PUSH(uint232)	pushes a 29-byte value onto the stack
7D	PUSH30	-	uint240	PUSH(uint240)	pushes a 30-byte value onto the stack
7E	PUSH31	-	uint248	PUSH(uint248)	pushes a 31-byte value onto the stack
7F	PUSH32	-	uint256	PUSH(uint256)	pushes a 32-byte value onto the stack
80	DUP1	value	value value	PUSH(value)	clones the last value on the stack

					the stack
82	DUP3	_ _ value	value _ _ value	PUSH(value)	clones the 3rd last value on the stack
83	DUP4	_ _ _ value	value _ _ _ value	PUSH(value)	clones the 4th last value on the stack
84	DUP5	... value	value ... value	PUSH(value)	clones the 5th last value on the stack
85	DUP6	... value	value ... value	PUSH(value)	clones the 6th last value on the stack
86	DUP7	... value	value ... value	PUSH(value)	clones the 7th last value on the stack
87	DUP8	... value	value ... value	PUSH(value)	clones the 8th last value on the stack
88	DUP9	... value	value ... value	PUSH(value)	clones the 9th last value on the stack
89	DUP10	... value	value ... value	PUSH(value)	clones the 10th last value on the stack
8A	DUP11	... value	value ... value	PUSH(value)	clones the 11th last value on the stack
8B	DUP12	... value	value ... value	PUSH(value)	clones the 12th last value on the stack
8C	DUP13	... value	value ... value	PUSH(value)	clones the 13th last value on the stack
8D	DUP14	... value	value ... value	PUSH(value)	clones the 14th last value on the stack
8E	DUP15	... value	value ... value	PUSH(value)	clones the 15th last value on the stack
8F	DUP16	... value	value ... value	PUSH(value)	clones the 16th last value on the stack
90	SWAP1	a b	b a	a, b = b, a	swaps the last two values on the stack
91	SWAP2	a _ b	b _ a	a, b = b, a	swaps the top of the stack with the 3rd last element
92	SWAP3	a _ _ b	b _ _ a	a, b = b, a	swaps the top of the stack with the 4th last element
93	SWAP4	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 5th last element
94	SWAP5	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 6th last element
95	SWAP6	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 7th last element
96	SWAP7	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 8th last element
97	SWAP8	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 9th last element
98	SWAP9	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 10th last element
99	SWAP10	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 11th last element
9A	SWAP11	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 12th last element
9B	SWAP12	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 13th last element

					with the 14th last element
9D	SWAP14	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 15th last element
9E	SWAP15	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 16th last element
9F	SWAP16	a ... b	b ... a	a, b = b, a	swaps the top of the stack with the 17th last element
A0	LOG0	offset length	-	LOG0(memory[offset:offset+length])	fires an event
A1	LOG1	offset length topic0	-	LOG1(memory[offset:offset+length], topic0)	fires an event
A2	LOG2	offset length topic0 topic1	-	LOG2(memory[offset:offset+length], topic0, topic1)	fires an event
A3	LOG3	offset length topic0 topic1 topic2	-	LOG3(memory[offset:offset+length], topic0, topic1, topic2)	fires an event
A4	LOG4	offset length topic0 topic1 topic2 topic3	-	LOG4(memory[offset:offset+length], topic0, topic1, topic2, topic3)	fires an event
A5	*Invalid*	-	-	-	-
A6	*Invalid*	-	-	-	-
A7	*Invalid*	-	-	-	-
A8	*Invalid*	-	-	-	-
A9	*Invalid*	-	-	-	-
AA	*Invalid*	-	-	-	-
AB	*Invalid*	-	-	-	-
AC	*Invalid*	-	-	-	-
AD	*Invalid*	-	-	-	-
AE	*Invalid*	-	-	-	-
AF	*Invalid*	-	-	-	-
B0	PUSH	-	-	???	???
B1	DUP	-	-	???	???
B2	SWAP	-	-	???	???
B3	*Invalid*	-	-	-	-
B4	*Invalid*	-	-	-	-
B5	*Invalid*	-	-	-	-
B6	*Invalid*	-	-	-	-
B7	*Invalid*	-	-	-	-
B8	*Invalid*	-	-	-	-
B9	*Invalid*	-	-	-	-
BA	*Invalid*	-	-	-	-
BB	*Invalid*	-	-	-	-
BC	*Invalid*	-	-	-	-
BD	*Invalid*	-	-	-	-
BE	*Invalid*	-	-	-	-
BF	*Invalid*	-	-	-	-
C0	*Invalid*	-	-	-	-
C1	*Invalid*	-	-	-	-
C2	*Invalid*	-	-	-	-
C3	*Invalid*	-	-	-	-
C4	*Invalid*	-	-	-	-
C5	*Invalid*	-	-	-	-
C6	*Invalid*	-	-	-	-
C7	*Invalid*	-	-	-	-
C8	*Invalid*	-	-	-	-
C9	*Invalid*	-	-	-	-
CA	*Invalid*	-	-	-	-
CB	*Invalid*	-	-	-	-
CC	*Invalid*	-	-	-	-
CD	*Invalid*	-	-	-	-
CE	*Invalid*	-	-	-	-
CF	*Invalid*	-	-	-	-
D0	*Invalid*	-	-	-	-
D1	*Invalid*	-	-	-	-
D2	*Invalid*	-	-	-	-
D3	*Invalid*	-	-	-	-
D4	*Invalid*	-	-	-	-
D5	*Invalid*	-	-	-	-
D6	*Invalid*	-	-	-	-
D7	*Invalid*	-	-	-	-
D8	*Invalid*	-	-	-	-
D9	*Invalid*	-	-	-	-
DA	*Invalid*	-	-	-	-
DB	*Invalid*	-	-	-	-
DC	*Invalid*	-	-	-	-
DD	*Invalid*	-	-	-	-
DE	*Invalid*	-	-	-	-
DF	*Invalid*	-	-	-	-
E0	*Invalid*	-	-	-	-
E1	*Invalid*	-	-	-	-
E2	*Invalid*	-	-	-	-

E5	*Invalid*	-	-	-	-
E6	*Invalid*	-	-	-	-
E7	*Invalid*	-	-	-	-
E8	*Invalid*	-	-	-	-
E9	*Invalid*	-	-	-	-
EA	*Invalid*	-	-	-	-
EB	*Invalid*	-	-	-	-
EC	*Invalid*	-	-	-	-
ED	*Invalid*	-	-	-	-
EE	*Invalid*	-	-	-	-
EF	*Invalid*	-	-	-	-
F0	CREATE	value | offset | length	addr	addr = new memory[offset:offset+length].value(value)	creates a child contract
F1	CALL	gas | addr | value | argsOffset | argsLength | retOffset | retLength	success	success, memory[retOffset:retOffset+retLength] = address(addr).call.gas(gas).value(value)(memory[argsOffset:argsOffset+argsLength])	calls a method in another contract
F2	CALLCODE	gas | addr | value | argsOffset | argsLength | retOffset | retLength	success	success, memory[retOffset:retOffset+retLength] = address(addr).callcode.gas(gas).value(value)(memory[argsOffset:argsOffset+argsLength])	???
F3	RETURN	offset | length	-	return memory[offset:offset+length]	returns from this contract call
F4	DELEGATECALL	gas | addr | argsOffset | argsLength | retOffset | retLength	success	success, memory[retOffset:retOffset+retLength] = address(addr).delegatecall.gas(gas)(memory[argsOffset:argsOffset+argsLength])	Homestead hardfork, EIP-7: calls a method in another contract, using the storage of the current contract
F5	CREATE2	value | offset | length | salt	addr	addr = new memory[offset:offset+length].value(value)	Constantinople harfork, EIP-1014: creates a child contract with a deterministic address
F6	*Invalid*	-	-	-	-
F7	*Invalid*	-	-	-	-
F8	*Invalid*	-	-	-	-
F9	*Invalid*	-	-	-	-
FA	STATICCALL	gas | addr | argsOffset | argsLength | retOffset | retLength	success	success, memory[retOffset:retOffset+retLength] = address(addr).staticcall.gas(gas)(memory[argsOffset:argsOffset+argsLength])	Byzantium hardfork, EIP-214: calls a method in another contract with state changes such as contract creation, event emission, storage modification and contract destruction disallowed
FB	*Invalid*	-	-	-	-
FC	*Invalid*	-	-	-	-
FD	REVERT	offset | length	-	revert(memory[offset:offset+length])	Byzantium hardfork, EIP-140: reverts with return data
FE	*Invalid*	-	-	-	-
FF	SELFDESTRUCT	addr	-	selfdestruct(address(addr))	destroys the contract and sends all funds to addr.